MAHEVE: An Efficient Reliable Mapping of Asynchronous Iterative Applications on Volatile and Heterogeneous Environments *

Raphaël Couturier, David Laiymani and Sébastien Miquée

University of Franche-Comté LIFC laboratory
IUT Belfort-Montbéliard, 2 Rue Engel Gros
BP 27 90016 Belfort, France
{raphael.couturier,david.laiymani,sebastien.miquee}@univ-fcomte.fr

Abstract. With the emergence of massive distributed computing resources, such as grids and distributed clusters architectures, parallel programming is used to benefit from them and execute problems of larger sizes. The asynchronous iteration model, called AIAC, has been proven to be an efficient solution for heterogeneous and distributed architectures. An efficient mapping of applications' tasks is essential to reduce their execution time. In this paper we present a new mapping algorithm, called MAHEVE (Mapping Algorithm for HEterogeneous and Volatile Environments) which is efficient on such architectures and integrates a fault tolerance mechanism to resist computing nodes failures. Our experiments show gains on a typical AIAC application execution time of about 55%, executed on distributed clusters architectures containing more than 400 computing cores with the JaceP2P-V2 environment.

1 Introduction

Nowadays, scientific applications require a great computation power to solve large problems. Though personal computers are becoming more powerful, in many cases they are not sufficient. One well adapted solution is to use computers clusters in order to combine the power of many machines. Distributed clusters form such an architecture, providing a great computing power, by aggregating the computation power of multiple clusters spread over multiple sites. Such an architecture brings users heterogeneity in computing machines as well as network latency. In order to use such an architecture, parallel programming is required. In the parallel computing area, in order to execute very large applications on heterogeneous architectures, iterative methods are well adapted[1,2].

These methods repeat the same instructions block until a convergence state and a desired approximation of the solution are reached. They constitute the only known approach to solving some kinds of problems and are relatively easy to parallelize. The Jacobi or the Conjugate Gradient[3] methods are examples of such methods. To parallelize them, one of the most used methods is the message passing paradigm which provides efficient mechanisms to exchange data

* This work was supported by the European Interreg IV From-P2P project.

between tasks. As such a method, we focus here on the asynchronous parallel iterative model, called AIAC[1] (for *Asynchronous Iterations – Asynchronous Communications*).

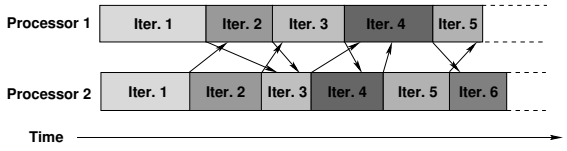

Fig. 1. Two processors computing in the Asynchronous Iterations – Asynchronous Communications (AIAC) model

In this model, as can be seen on Figure 1, after each iteration, a task sends its results to its neighbors and immediately starts the next iteration with the last received data. The receiving and sending mechanisms are asynchronous and tasks do not have to wait for the reception of dependency messages from their neighbors. Consequently, there is no idle time between two iterations. Furthermore, this model is tolerant to messages loss and even if a task is stopped the remaining tasks continue the computation, with the last available data. Several experiments[2] show the relevance of the AIAC algorithms in the context of distributed clusters with high latency between clusters. These works underline the good adaptability of AIAC algorithms to network and processor heterogeneity.

In a previous study[4] we proposed the implementation of two static mapping algorithms of tasks to processors dedicated to the AIAC model on heterogeneous distributed clusters. Both these two algorithms, AIAC-QM (for *AIAC Quick-quality Map*) and F-EC (for *Farhat Edges-Cuts*) showed an important performances improvement by significantly reducing the application execution time. These experiments were performed by using the fully fault tolerant JaceP2P-V2 environment, described in next section. In our previous experiments we did not introduce computing nodes failures during the computation. As architecture heterogeneity continually evolves according to computing nodes volatility, we have to take care more precisely about the heterogeneity of the target platform. Thus in this paper our main contribution is to propose a new mapping algorithm called MAHEVE (*Mapping Algorithm for HEterogeneous and Volatile Environments*). This algorithm explicitly tackles the heterogeneity issue and introduces a level of dynamism in order to adapt itself to the fault tolerance mechanisms. Our experiments show gains up to 65% on application execution time, with faults during executions, which is about 10 points better than AIAC-QM and about 25 points better than F-EC, and MAHEVE also outperforms them in experiments with no fault during executions.

The rest of this paper is organized as follows. Section 2 presents the JaceP2P-V2 middleware by describing its architecture and briefly presenting its fault tolerance mechanisms. Section 3 formalizes our mapping and fault tolerance problems and quotes existing issues to address them. Section 4 describes the new mapping strategy we propose, MAHEVE. In Section 5 we present the experiments we conducted on the Grid'5000 testbed with more than 400 computing cores. Finally, we give some concluding remarks and plan our future work in Section 6.

2 JaceP2P-V2

JaceP2P-V2[5] is a distributed platform implemented in Java, dedicated to developing and executing parallel iterative asynchronous applications. It is fully fault tolerant allowing it to execute parallel applications over volatile environments. To our knowledge this is the only platform dedicated to designing and executing AIAC algorithms in such volatile environments.

The JaceP2P-V2 platform part, which is based on the daemons and supervisors paradigm, is composed of three main entities:

- The "super-nodes", which are in charge of supervising free computing nodes connected to the platform;
- The "spawner", which is launched by a user wanting to execute a parallel application. It is in charge of a group of computing nodes and monitors them. If one of them fails, it requires a replacing one to a super-node;
- The "daemon", first connects to a super-node and waits for a task to execute. Each daemon can communicate directly with its computing neighbors.

To be able to execute AIAC applications, JaceP2P-V2 has an asynchronous messaging mechanism, and to resist daemons failures, it implements a checkpoint/restart mechanism by using a distributed backup mechanism called the *uncoordinated distributed checkpointing*[6]. This decentralized procedure allows the platform to be very scalable, with no weak points and does not require a secure nor a stable station for backups. When a daemon dies, it is replaced by another one, as we suppose that there are enough available free nodes. Moreover, to resist supervisors failures and for scalability, some extra nodes are reserved. For more details on the JaceP2P-V2 platform, interested readers can refer to [5].

3 Mapping and fault tolerance problems

3.1 Model formalization

Application modeling The TIG[7] (*Task Interaction Graph*) model is the most appropriate to our problem, as it only models relationships between tasks. In this model, all the tasks are considered simultaneously executable and communications can take place at any time during the computation, with no precedence nor synchronization. As a reminder, during an iteration in the AIAC model, each task computes its job and sends its results to its neighbors, and immediately starts the next iteration.

In the TIG model, a parallel application is represented by a graph $GT(V, E)$, where $V = \{V_1, V_2, \dots V_v\}$ is the set of $|V|$ vertices and $E \subset V \times V$ is the set of undirectional edges. The vertices represent tasks and the edges represent the mutual communication among tasks. A function $EC : V \to \mathbb{R}^+$ gives the computation cost of tasks and $CC : E \to \mathbb{R}^+$ gives the communication cost for message passing on edges. We define $|V| = v$, $EC(V_i) = e_i$ and $CC(V_i, V_j) = c_{ij}$. Another function $D : V \to \mathbb{N}^+$ gives the amount of dependencies of a task, and we define $D(V_i) = d_i$.

Architecture modeling A distributed clusters architecture can be modeled by a three-level-graph. The levels are *architecture* (a) (here the Grid'5000 grid), *cluster* (c), and *computing node* (n) levels. Let $GG(N, L)$ be a graph representing a distributed clusters architecture, where $N = \{N_1, N_2, \dots N_n\}$ is the set of $|N|$ vertices and L is the set of $|L|$ undirectional edges. The vertices represent the computing nodes and the edges represent the links between them. An edge $L_i \in L$ is an unordered pair $(N_x, N_y) \in N$, representing a communication link between nodes N_x and N_y. A function $WN : N \to \mathbb{R}^+$ gives the computational power of nodes and another function $WL : L \to \mathbb{R}^+$ gives the communication latency of links. We define $WN(N_i) = wn_i$ and $WL(L_i, L_j) = wl_{ij}$. Let be $|C|$ the number of clusters contained in the architecture. A function $CN : C \to \mathbb{N}^+$ gives the amount of computing nodes contained in a cluster, and another function $CF : C \to \mathbb{N}^+$ gives the amount of available computing nodes (not involved in computation) of a cluster. We define $CN(C_i) = C_{Ni}$ and $CF(C_i) = C_{Fi}$. We also define $C_{\overline{P}fi}$ as the average power of available resources of cluster C_i.

We evaluate the *heterogeneity degree* of the architecture, noted hd, by using the *relative standard deviation* method, with $hd = \frac{\sigma_{PN}}{avg_{PN}}$ where avg_{PN} is the average computing power of nodes and σ_{PN} represents the standard deviation of computing nodes power. This measure provides us the coefficient of variation of the platform in percentage – we only consider $0 \leq hd \leq 1$ as considering values of $hd > 1$ is not relevant, as $hd = 1$ denotes a fully heterogeneous platform.

Mapping functions When a parallel application *App*, represented by a graph GT, is mapped on a distributed clusters architecture, represented by a graph GG, the execution time of the application, $ET(App)$, can be defined as the execution time of the slowest task. Indeed, an application ends when all the tasks have detected convergence and reached the desired approximation of the solution. We define $ET(App) = \max_{i=1\dots v}(ET(V_i))$, where the execution time of each task i $(i = 1 \dots v)$, $ET(V_i)$, is given by $ET(V_i) = \frac{e_i}{wn_i} + \sum_{j \in J} c_{ij} \times wl_{ij}$ where e_i is the computational cost of V_i, wn_i is the computational power of the node N_i on which V_i is mapped, J represents the neighbors set of V_i, c_{ij} is the amount of communications between V_i and V_j, and wl_{ij} is the link latency between the computing nodes on which V_i and V_j are mapped. As described in this formula, the execution time of a task depends on the task weight and on the communications which may occur between this task and its neighbors. We underline here that in the AIAC model, it is impossible to predict the number of iterations of a task. So it is difficult to evaluate a priori its cost e_i.

This tasks mapping problem is similar to the classical graph partitioning and task assignment problem, and is thus NP-complete.

3.2 Fault tolerance

In volatile environments, computing nodes can disconnect at any time during the computation, and have thus to be efficiently replaced. The replacing nodes should be the best ones at the fault time, by searching them in available nodes. As

executing environments can regularly evolve, due to computing nodes volatility, a mapping algorithm has to keep a correct overview of the architecture, in real time. Thus, criteria to assign tasks to nodes should evolve too.

Another problem appears after multiple crashes: some tasks may have migrated over multiple computing nodes and clusters, and the initial mapping may be totally changed. So, after having suffered some nodes failures the tasks mapping could not always satisfy the mapping criteria (not on the more powerful available machine, too far away from its neighbors...). A good fault tolerance policy has to evolve dynamically with the executing environment.

3.3 Specificities of the AIAC mapping problem

An important point to take into consideration is that we do not allow the execution of multiple tasks on the same computing node, as this provides a fall of performances when this one fails. Indeed we should redeploy all of the tasks from this node to another one, using last saves, which can be spread on multiple computing nodes. This may result in large communication overheads and in a waste of computation time. Nevertheless, to benefit from multi-cores processors, we use a task level parallelism by multi-threaded sequential solvers for example.

Another important point in the AIAC model is that as the JaceP2P-V2 environment is fault tolerant and tasks save checkpoints on their neighbors, it is more efficient to save on near nodes than on far ones in order to reduce the communication overhead during this operation, and to restart a task faster.

3.4 Related work

In the literature of the TIG mapping many algorithms exist, which can be broadly classified into two categories. The first one is the *Edge-cuts optimization* class, which minimizes the use of the penalizing links between clusters. As tasks are depending on neighbors, which are called dependencies, the goal is to choose nodes where distance, in term of network, is small to improve communications between tasks. Here we can cite the Farhat's algorithm[8], and Metis[9] and Chaco[10] which are libraries containing such kind of algorithms. The second category is the *Execution time optimization* class, which aims at minimizing the whole application execution time. These algorithms look for nodes which can provide the smallest execution time of tasks using their computational power. Here we can cite QM[11], FastMap[12], and MiniMax[13] as such kind of algorithms.

Both classes of algorithms may fit with our goals as in our model we have both the computational power of nodes and communication costs which may influence the applications performances.

All mentioned algorithms do not tackle the computing nodes failures issue, or only basically by applying the same policy. As explained in Section 3.2, a more efficient and dedicated replacement function is needed. Nevertheless, to the best of our knowledge, no tasks mapping algorithm, addressing explicitly both the executing platform heterogeneity and the computing nodes failures issues, exists.

4 MAHEVE

Here we present our new tasks mapping strategy, called MAHEVE (for *Mapping Algorithm for HEterogeneous and Volatile Environments*). This algorithm aims at taking the best part of each category mentioned in Section 3.4, the edge-cuts minimization and the application execution time optimization algorithms.

This new algorithm can be divided into two parts. The first part aims at performing the initial mapping, and the second part is devoted to search replacing nodes when computing nodes failures occur.

4.1 Initial mapping

In this section we will study the main mechanisms of the *static mapping* done by MAHEVE, which is composed of three phases: sort of clusters, sort of tasks, and the effective mapping, which maps tasks (in their sort order) on nodes of clusters (also in their sort order) with a reservation of some nodes in each cluster.

Sorting clusters The first step of the initial mapping is to sort clusters according to the executing platform heterogeneity degree hd. The main principles are that a cluster obtains a better mark when $hd < 0.5$ and it contains more computing nodes than other clusters (C_{Fi}, the number of available free nodes, is privileged), and when $hd \geq 0.5$ and it contains more powerful computing nodes ($C_{\overline{P}fi}$, the average free computation power, is privileged). These choices come from several experiments with the AIAC model, which show that in such environments it is more efficient to privilege the computation power or the number of nodes. As the number of nodes, C_{Fi}, and the average free computing power, $C_{\overline{P}fi}$, are not in the same order of magnitude, we normalize them with two functions, $normN$ and $normP$. We note $normN\,(C_{Fi}) = NC_{Fi}$ and $normP(C_{\overline{P}fi}) = NC_{\overline{P}fi}$.

The formula used to give a mark, M_i, to a cluster is

$$M_i = {NC_{\overline{P}fi}}^{hd} + {NC_{Fi}}^{1-hd}. \tag{1}$$

This compromise function allows us to privilege clusters following our criteria, as explained previously, according to the heterogeneity degree. If we study its limits for the hd's extremities, $hd = 0$ and $hd = 1$, we obtain $\lim_{hd\to 0} M_i = NC_{Fi} + 1$ and $\lim_{hd\to 1} M_i = NC_{\overline{P}fi} + 1$, which fit with our objectives.

Clusters are so sorted and placed in a list containing them, starting from the cluster which receives the better mark to the one which receives the lower mark.

Sorting tasks Like clusters, tasks are also sorted according to the heterogeneity degree of the executing platform. This sorting is done in the same way as previously, as when $hd < 0.5$ tasks with higher dependencies will be privileged, and when $hd \geq 0.5$ tasks with higher computing cost are privileged.

The main function used to classified tasks is

$$Q_i = {e_i}^{hd} \times {d_i}^{1-hd} \tag{2}$$

where Q_i is the evaluation of the task i according to the heterogeneity degree hd and d_i, the amount of dependencies of task i.

Tasks are taken in the order of the first sort, determined with equation (2), and each task is placed in a new list (the final one) and some of its dependencies are added. We note $Nb_i = {d_i}^{1-hd}$ this amount of dependencies as the lower the heterogeneity degree is the higher this number will be. This final operation allows to control the necessary locality of tasks according to hd.

Mapping method The third step of the initial mapping is to allocate tasks to nodes. As clusters and tasks have been sorted accordingly to the executing platform heterogeneity degree, ordered from the highest mark to the lowest, this function maps tasks on almost all available computing nodes of clusters, in their respective order in lists (for example a task classified first in the tasks list is mapped on an available node of the cluster classified first in the clusters list). The idea here is not to fulfill each cluster, but to preserve some computing nodes in each cluster. These conserved nodes will be used to replace failed nodes.

Here we can mentioned that the whole mapping process (the three steps) has a complexity of $O(|V|log(|V|))$, where |V| is the number of tasks.

4.2 Replacing function

As shown in the previous section, during the initial mapping some computing nodes in each cluster have been preserved. When a node fails this function replace it by a free node of the same cluster. If none is available this function sorts again clusters, to take into consideration platform modifications, and replace the failed node by one available in the new sorted clusters list. This mechanism allows to keep tasks locality and a real time overview of the executing platform.

5 Experimentation

5.1 A typical AIAC application and the execution platform

We used the "Kernel CG" application of the NAS Parallel Benchmarks (NPB) [14] to evaluate the performances of our new mapping algorithm. This benchmark is designed to be used on large architectures, as it stresses communications, by processing unstructured matrix vector multiplication with a Conjugate Gradient method. As this method cannot be executed with the asynchronous iteration model we have replaced it by another method called the multisplitting method, which supports the asynchronous iterative model. More details about this method can be found in [1]. The chosen problem used a matrix of size $5,000,000$ with a low bandwidth, fixed to $35,000$. This bandwidth size generates, according to the problem size, between 8 and 20 neighbors per tasks. This application was executed on 64 nodes.

The platform used for our tests, called Grid'5000[15], is a French nationwide experimental set of clusters which provides us with distributed clusters architectures (28 heterogeneous clusters spread over 9 sites). We used three distributed

clusters architectures, each having a different heterogeneity degree. The first one was composed of four clusters spread over four sites, with a total of 106 computing nodes representing 424 computing cores with $hd = 0.08$; the second one was composed of four clusters spread over three sites, with a total of 110 computing nodes representing 440 computing cores with $hd = 0.50$; and finally the third one was composed of five clusters spread over four sites with 115 computing nodes representing 620 computing cores with $hd = 0.72$.

All computing nodes of these clusters have at least 4 computing cores (in the last used architecture, with $hd = 0.72$, two clusters are composed of 8 computing cores machines) with a minimum of 4GB of memory (in order to execute the application with a big problem size). All computing nodes can communicate with each other through an efficient network. Nevertheless, this latter is shared with many other users so high latencies appear during executions.

5.2 Experiments

During executions, we introduced two failures in computing nodes involved in the computation every 20 seconds to simulate a volatile environment. Unfortunately, we did not have the opportunity to realize experiments with more computing nodes over more sites with problems of larger sizes, but we plan to extend our experiments in the future.

Here we present the results of the evaluation of the MAHEVE algorithm, compared with FT-AIAC-QM (for *Fault Tolerant AIAC-QM*) and FT-FEC (for *Fault Tolerant F-EC*) which are respectively the fault tolerant versions of the AIAC-QM and F-EC mapping algorithms presented in [4]. Table 1 shows the execution times of each mapping algorithm compared to the default mapping strategy of the JaceP2P-V2 platform, with the corresponding gains on application execution time, given in brackets. It presents both the executions with faults (WF) and the fault free (FF) executions.

hd	Default		FT-AIAC-QM		FT-FEC		MAHEVE	
	FF	WF	FF	WF	FF	WF	FF	WF
0.08	80	229	63 (21%)	178 (22%)	61 (23%)	154 (33%)	60 (25%)	113 (50%)
0.50	67	242	61 (9%)	118 (51%)	63 (6%)	133 (45%)	54 (20%)	85 (65%)
0.72	67	192	59 (12%)	99 (45%)	65 (3%)	121 (33%)	52 (22%)	86 (53%)

Table 1. Application execution time in seconds and corresponding gains on various platforms using different mapping algorithms, with fault free (FF) executions and with 2 node failures each 20 seconds (WF) executions.

First of all, we can note that all mapping algorithms provide an enhancement of the application performances by considerably reducing its execution time, especially for executions with node failures, with an average gain of about 45% in general in comparison to the default policy. If we focus on executions with node

failures (WF), FT-FEC is efficient on architectures with a low heterogeneity degree ($hd = 0.08$) by providing gains of about 33%, and gains are roughly the same on heterogeneous architectures ($hd = 0.72$). FT-AIAC-QM is efficient on architectures with a high heterogeneity degree ($hd = 0.72$) by providing gains of about 45%, whereas it is not so efficient on homogeneous architectures ($hd = 0.08$) by providing gains of about 22%. We can note here that on an architecture with a heterogeneity degree of 0.50 FT-AIAC-QM is more efficient than FT-FEC by providing gains up to 50%. Here we point out that in fault free executions (FF), both algorithms also provide gains on their respective favorite architectures, though gains are less great than in executions with faults (WF).

Now if we focus on the performances of our new solution MAHEVE, we can see that it is all the time better than other algorithms. As can be seen in Table 1, in executions with faults (WF), it reduces the application's execution time by about 50% on homogeneous architectures (here of 0.08 heterogeneity degree) which is more than 25 point better than FT-FEC and near 30 points better than FT-AIAC-QM. On heterogeneous architectures (here of 0.72 heterogeneity degree) it also outperforms other mapping algorithms by reducing the application execution time by about 53% which is almost 10 points better than FT-AIAC-QM and 20 points better than FT-FEC. On middle heterogeneity degree architectures (here of 0.50 heterogeneity degree), MAHEVE is once again better than its two comparative mapping algorithms by reducing the application execution time by about 53%. These good performances come from the fact that it is designed to be efficient on both architectures, homogeneous and heterogeneous. Moreover, as it integrates a fault tolerance *security* in the initial mapping, it is more efficient when computing nodes fail. Here we can point out that this algorithm allows in general gains on application execution time of about 55%. In fault free executions (FF), it outperforms once again the two other algorithms.

6 Conclusion and future works

In this paper we have presented a new mapping algorithm, called MAHEVE, to address the AIAC mapping issue on heterogeneous and volatile environments. It aims at doing an efficient mapping of tasks on distributed clusters architectures by taking the best part of the two known approaches, application execution time optimization and edge-cuts minimization. Experiments, though using a single application, show that it is the most efficient mapping algorithm on all kinds of architectures, as it takes into account their heterogeneity degree and adapt its sort methods to it. We have shown that it is all the time better than the two other comparative mapping algorithms, FT-AIAC-QM and FT-FEC. This can be explained by the fact that it not only takes care about computing nodes and clusters, but also about the tasks' properties, what refines the mapping solution.

In our future works we plan to enhance the MAHEVE algorithm performances by modifying the notation of clusters, since their locality has not yet been taken into consideration. This would favor tasks locality, which would reduce communications delays and provide a much better convergence rate. We also have to validate the algorithm performance with other AIAC applications.

Acknowledgement

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed[15], being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies.

References

1. J. Bahi, S. Contassot-Vivier, and R. Couturier. *Parallel Iterative Algorithms: from Sequential to Grid Computing*, volume 1 of *Numerical Analysis & Scientific Computating*, chapter Asynchronous Iterations. Chapman & Hall/CRC, 2007.
2. J. Bahi, S. Contassot-Vivier, and R. Couturier. Performance comparison of parallel programming environments for implementing AIAC algorithms. *Journal of Supercomputing*, 35(3):227–244, 2006.
3. J. K. Reid. *On the method of conjugate gradients for the solution of large sparse systems of linear equations*, pages 231–254. Academic Press Inc, March 1971.
4. R. Couturier, D. Laiymani, and S. Miquée. Mapping asynchronous iterative applications on heterogeneous distributed architectures. In *PDSEC*, Atlanta, USA, 2010. IEEE Computer Society Press.
5. J.-C. Charr, R. Couturier, and D. Laiymani. Jacep2p-v2: A fully decentralized and fault tolerant environment for executing parallel iterative asynchronous applications on volatile distributed architectures. In *GPC*, pages 446–458, 2009.
6. E. N. Elnozahy, L. Alvisi, Y. Wang, and D. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.
7. D. L. Long and L. A. Clarke. Task interaction graphs for concurrency analysis. In *ICSE*, pages 44–52, 1989.
8. C. Farhat. A simple and efficient automatic fem domain decomposer. *Computers & Structures*, 28(5):579 – 602, 1988.
9. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.
10. B. Hendrickson and R. W. Leland. *The Chaco User's Guide*. Sandia National Laboratory, Albuquerque, 1995.
11. P. Phinjaroenphan. *An Efficient, Pratical, Portable Mapping Technique on Computational Grids*. PhD thesis, School of Computer Science and Information technology Science, Engineering and Technology Portfolio, RMIT University, 2006.
12. S. Sanyal, A. Jain, S. K. Das, and Rupak Biswas. A hierarchical and distributed approach for mapping large applications to heterogeneous grids using genetic algorithms. In *CLUSTER*, pages 496–499, 2003.
13. S. Kumar, S. K. Das, and R. Biswas. Graph partitioning for parallel applications in heterogeneous grid environments. In *IPDPS*, 2002.
14. D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagun, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. venkatakrishnan, and S. Weeratunga. The NAS Parallel Benchmarks. Technical Report RNR-94-007, NASA Advanced Supercomputing (NAS) Division, March 1994.
15. Grid'5000. http://www.grid5000.fr.